社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
IT2002-25 (2002-07)

最頻 Digram 統合に基づくデータ圧縮法

神田 勝† 石田 崇† 小林 学†† 平澤 茂一†

† 早稲田大学理工学部経営システム工学科 〒169-8555 東京都新宿区大久保 3-4-1
†† 湘南工科大学工学部情報工学科 〒251-8511 神奈川県藤沢市辻堂西海岸 1-1-25
E-mail: †{kanda,ishida,hirasawa}@hirasa.mgmt.waseda.ac.jp, ††kobayasi@info.shonan-it.ac.jp

あらまし 近年，Sequitur アルゴリズムや MPM 符号等のデータ系列の文法を用いたデータ圧縮法の研究が盛んに行われている．文法を用いた圧縮法では，データ系列を直接符号化するのではなく，データ系列を生成するような文法を構成し，それを符号化する．Sequitur アルゴリズムでは，Digram と呼ばれる 2 記号の並びを単位とし，データ系列の前方一致検索で一致する Digram から文法を生成するアルゴリズムである．これに対して，中村らはデータ系列中の最頻の Digram から文法を生成するアルゴリズムを提案している．本研究では，中村らのアルゴリズムと同様最頻の Digram によって文法を生成するが，この文法の符号化に算術符号を適用することを前提とする．このとき符号化するデータ系列に対して本手法を適用し，算術符号の理想符号長を計算しながら符号長の観点から最適な文法を生成する手法を提案する．またカルガリーデータに対して本手法を適用し，シミュレーションによって評価を行う．更に N 記号のデータに対し提案手法の計算量及びメモリ量が Sequitur アルゴリズムと同様 $O(N)$ であることを示す．

キーワード 情報源符号化，無歪み圧縮，文法，Digram，算術符号

Data Compression Algorithm Based on Unification of the Most Frequent Digram

Masaru KANDA†, Takashi ISHIDA†, Manabu KOBAYASHI††, and Shigeichi HIRASAWA†

† Dept. of Industrial and Management Systems Engineering
Waseda University
3-4-1 Ohkubo, Shinjuku-ku,Tokyo 169-8555
†† Dept. of Information Science, Shonan Institute of Technology
1-1-25 Nishikaigan Tsujidoh Fujisawa-si Kanagawa 251-8511
E-mail: †{kanda,ishida,hirasawa}@hirasa.mgmt.waseda.ac.jp, ††kobayasi@info.shonan-it.ac.jp

Abstract Recently, the compression algorithms based on the grammar have been proposed. In Sequitur algorithm, if the sequence has two same Digrams, the Digram is registered as a rule. While Nakamura et al. have proposed an algorithm which the most frequent Digram is registered as a rule. In this paper, we propose an algorithm that we aplly the adaptive arithmetic code to the sequence constructed by Nakamura's algorithm. Moreover we make the grammar expected to shorten the code length from the sequence by calculating the code length of the arithmetic code.

Key words Data Compression, Lossless Compression, grammar, Digram, arithmetic code

1. はじめに

確率構造が未知の情報源から出力されるデータ系列に対するデータ圧縮法として，文法に基づく手法が近年盛んに研究されている [5] [6] [7]．文法に基づく圧縮法ではデータ系列を直接符号化するのではなく，データ系列を生成する文法を作成し，それを符号化する．もし，データ系列に対して簡潔な文法を与えることができれば，元のデータ系列を符号化するよりも文法を符号化することによってより良い圧縮率が達成される．

Craig G.Nevill-Mannning と Ian H.Witten により提案された Sequitur アルゴリズム [3] は，文字列や単語から階層的な文法規則を抽出するアルゴリズムである．この Sequitur アルゴリズムによって階層化された文法と算術符号を組み合わせたデータ圧縮法が提案され [4]，その簡便な手法にも関わらず非常に良好な圧縮率を達成している．具体的には Sequitur アルゴリズムは Digram と呼ばれる 2 記号の並びを単位とし，データ系列の前方一致検索において一致する Digram から文法を生成するアルゴリズムである．

これに対して，中村らはデータ系列中の最頻の Digram から文法を生成する手法を提案している [1]．Sequitur アルゴリズム及び中村らの手法は文法を生成する手法に違いはあるものの，Digram をルールとして登録し，新たな別の記号で置き換えるという点で類似している．

本研究では，中村らのアルゴリズムと同様最頻 Digram に

よって文法を生成するが，これに対して算術符号を適用することを考える．このとき，符号化する文法に対して算術符号の理想符号長を計算しながら，符号長の観点から最適な文法を生成する手法を提案する．本手法をカルガリーデータに対するシミュレーションによって，その結果を考察する．また，入力されるデータの長さを N とすると提案手法は計算量及びメモリ量が $O(N)$ であることを示す．

2. 準　　備

2.1 文　法[11]

まず初めに文法のための基本的な準備をする．文法は以下のような 4 項組 $G=(\Sigma_T,\Sigma_N,R,S)$ で定義される．

（1） Σ_T：終端記号の有限集合．

（2） Σ_N：非終端記号の有限集合．

（3） R：文の生成規則の有限集合（グラマー）．

（4） $S(\in\Sigma_N)$：開始記号．

ただし $\Sigma_T\cap\Sigma_N=\phi$ とする．慣例として終端記号を小文字のアルファベット，非終端記号を大文字のアルファベットで表記することとする．このとき文の生成規則を以下のように定義する．

$$\mu\rightarrow\omega \qquad \mu\in\Sigma_N \qquad \omega\in(\Sigma_N\cup\Sigma_T)^*$$

以下では文の生成規則をルールと呼び，特定の文法 G におけるルールの集合をグラマーと呼ぶ．文法に基づく符号では，Σ_T が情報源アルファベットに相当し，データ系列 x は $x\in\Sigma_T^+$ となる．また，$\Sigma=\Sigma_T\cup\Sigma_N$ は記号全体の集合を表し，全順序関係が定義されているとする．

2.2 Sequitur アルゴリズム[3]

データ系列 x において，連続する 2 記号の並びを Digram と呼ぶ．Sequitur アルゴリズムはデータ系列 x を逐次的に読み込み，Digram を単位として，グラマーを生成する．その際，以下の 2 つの条件を満足する．

(I) Digram Uniqueness

グラマー中に同一の Digram が 2 つ以上存在してはならない．

(II) Rule Utility

ルールはグラマー中で 2 回以上使用されなくてはならない．

Sequitur では制約 (I) を満足するために，読み込まれた Digram が以前現れている Digram と一致した場合，左辺を未使用の非終端記号，右辺を一致した Digram とする新しいルールを生成する．更に一致した 2 つの Digram をこの非終端記号で置き換える．置き換えられた非終端記号も終端記号と同様に 1 記号と見なす．

また，制約 (II) を満足するために，グラマー中で 1 度しか用いられていないルールを削除し，そのルールを表す非終端記号を，ルールを構成している記号列で置き換える．この，ルールを削除し置き換える作業によって，ルールの右辺が 3 記号以上となるルールが生成される．

以下に Sequitur アルゴリズムの詳細な流れを示す．

＜Sequitur アルゴリズム＞

① $S\rightarrow\lambda$ (注1) というルールを生成する．

② データ系列から 1 記号を読み込みルール S の右辺に追加し現在の記号とする．読み込むべき記号がなければ終了．

(注1)：λ は空列を表す．

表 1　Sequitur アルゴリズムの動作例

No.	symbol	String	Grammar	備考
1	a	a	$S\rightarrow a$	
2	b	ab	$S\rightarrow ab$	
3	c	abc	$S\rightarrow abc$	
4	a	$abca$	$S\rightarrow abca$	
5	b	$abcab$	$S\rightarrow abcab$	ab が一致
			$S\rightarrow AcA$	Rule A 生成
			$A\rightarrow ab$	
6	c	$abcabc$	$S\rightarrow AcAc$	Ac が一致
			$A\rightarrow ab$	
			$S\rightarrow BB$	Rule B 生成
			$A\rightarrow ab$	A が 1 度
			$B\rightarrow Ac$	のみ使用
			$S\rightarrow BB$	Rule A 削除
			$B\rightarrow abc$	
7	b	$abcabcb$	$S\rightarrow BBb$	
			$B\rightarrow abc$	
8	c	$abcabcbc$	$S\rightarrow BBbc$	bc が一致
			$B\rightarrow abc$	
			$S\rightarrow BBC$	Rule C 生成
			$B\rightarrow aC$	
			$C\rightarrow bc$	
9	a	$abcabcbc$	$S\rightarrow BBC$	終了
			$B\rightarrow aC$	
			$C\rightarrow bc$	

③ 現在の記号と直前の記号で生成される Digram がグラマー中に存在しているか検索する．もし存在しているならば ④ へ．そうでなければ ② へ．

④ 一致した Digram が含まれるルールの右辺が 3 記号以上のもの，2 記号のもので場合分けを行う．

(1) 3 記号以上の場合

一致した Digram を新しいルールの右辺，未使用の非終端記号を左辺とする新しいルールを作り，同一の Digram が出現した 2 箇所を，そのルールを表す非終端記号で置き換える．

(2) 2 記号の場合

そのルールを表す非終端記号でルール S の右辺の Digram を置き換える．

⑤ もし置き換えられた Digram 中に非終端記号が存在し，かつその記号が 1 箇所にのみ存在するときはそのルールを削除する．更にその非終端記号をそのルールの右辺の記号列で置き換える．

⑥ 置き換えた非終端記号も終端記号と同様に扱い，それを現在の記号として ③ へ．　□

具体的なデータ系列（$x=abcabcbc$）に対する Sequitur アルゴリズムの動作例を表 1 に示す．

2.3 高頻度隣接の連結に基づいたグラマー生成法[1][2]

Sequitur はデータ系列から逐次的にルールを生成していたが，中村らは最も頻度の高い Digram からルールを生成する手法を提案した．この手法では，多く出現する Digram をルールとすれば，頻繁に出現するパターンをルールとすることが可能となる．

＜高頻度隣接の連結に基づく手法＞

① データ系列 x をルール S の右辺とする．S の右辺について各 Digram の頻度を調べる．

② ルール S の右辺中の各 Digram の出現頻度が全て 1 ならば終了．そうでなければ最頻の Digram を右辺とする新たな

表 2　中村らのアルゴリズムの動作例

No.	Grammar	Digram	頻度	備考
1	$S \rightarrow abcabcbcabc$	ab	3	bc が最頻
		bc	4	
		ca	2	
		cb	1	
2	$S \rightarrow aAaAAaA$	aA	3	Rule A 作成
	$A \rightarrow bc$	Aa	2	
		AA	1	aA が最頻
3	$S \rightarrow BBAB$	BB	1	Rule B 作成
	$A \rightarrow bc$	BA	1	
	$B \rightarrow aA$	AB	1	全て頻度が 1，終了

ルールを生成する．このとき，ルール S の対象となった最頻の Digram は新たなルールの左辺の非終端記号で置き換える．

③　置き換えた非終端記号も終端記号と同様に扱い，その直前および直後の記号との Digram の頻度を求め ② へ．　□

具体的なデータ系列（$x = abcabcbcabc$）に対する中村らのアルゴリズムの動作例を表 2 に示す．

3. 符号化法

3.1 データの 1 次元化

前節で示した Sequitur アルゴリズムや中村らの手法によって生成されたグラマーはルールの集合であるのでグラマーを符号化するためにはこれを 1 次元化する必要がある．1 次元化されたグラマーをグラマー記号列と呼ぶことにする．1 次元化する手法として以下の 2 つが挙げられる．

(1)　全てのルールを連接する手法

生成された各ルールに対し，区切り記号を挿入しながら連接する手法である．

(2)　ルール S 内で他のルールを参照する手法

中村らはルールである箇所を示すバイナリーのデータ系列を用意する手法 [1] [2] を用いている．

全てのルールを連接する代わりに，ルール S の右辺において，S 以外のルールも参照できるように構成し直す．すなわち，ルール S の右辺を左から読んだときに，初めて出てくる非終端記号はその非終端記号が表すルールの右辺の記号列で置き換えるのである．このとき，この手順によって置き換えられた既出の非終端記号及び終端記号の列が明示される必要があるが，これは置き換える手順が木構造で表現できることを利用し，木構造を表すバイナリーのデータ系列を生成することによって可能となる．

具体的にはルール S の右辺を左から読んだときに初めて出現する非終端記号を根ノードとし，そのルールの右辺の Digram の記号をそれぞれ子ノードとして展開する．このとき，展開された子ノードが，初めて出現する非終端記号である場合にはこれを再び親ノードとして子ノードを展開し，ルール S 中の初めて出現する全ての非終端記号が展開されるまでこれを繰り返す．次にルール S において木の根ノードにあたる非終端記号をその木の葉ノードになっている記号列で置き換え，この系列を改めて S' と表す．更に以下の手順に従ってこの木構造を表すバイナリー系列を生成する．まず S の右辺の系列に対し，木構造となっているとき根ノード及び中間ノードには "1" を割り振り，葉ノード及び木構造となっていない終端記号，非終端記号には "0" を割り振る．次に S の右辺を左から見てゆき，深さ優先の順に割り振ったビットを並べる．こうしてできたビット系列を T と表す．結果的に S' 及び T を符号語とする．図 1 に具体例を示す．

実際に符号化する際には，復号器側でもルールを参照できるようにルール S の右辺の記号列を左から読んだときに先に現れるルールに対して小さい番号を付加し，この順序に従うように非終端記号を改めて付け直す(注2)

$$x = abcdabcaaaaaaab$$
$$S \rightarrow BdBCCCA$$
$$A \rightarrow ab$$
$$B \rightarrow Ac$$
$$C \rightarrow aa$$
$$\Downarrow$$
$$S' = abcdBaaCCA$$
$$T = 1100000100000$$

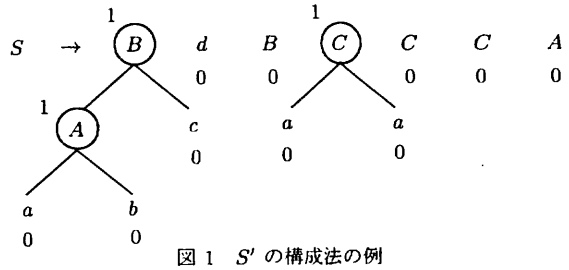

図 1　S' の構成法の例

3.2 符号化

1 次元化されたグラマーをグラマー記号列と呼ぶ．グラマー記号列を以下の 2 つの手法により符号化する．

(1)　グラマー記号列に対しては符号長 $\log_2 C$　(C：グラマー記号列中に出現する記号数) で符号を割り当て，また，バイナリーのデータもそのまま送る手法．

(2)　グラマー記号列には動的算術符号，バイナリーのデータに対してはランレングス符号を用いる手法．

4. 提案手法

4.1 従来法の問題点および提案の概要

Sequitur アルゴリズムのルール生成手法は，データ系列を逐次的に処理するため，偶然 2 回 Digram が出現した場合でもルールとしてしまう．一度ルールが生成されると，そのルールがあまり出現しないにも関わらず削除されないまま残るため，その後のルール生成に悪影響を及ぼしてしまう場合がある．一方，中村らのアルゴリズムでは，一度データ系列を全て読み込み，最も出現頻度の高い Digram から順にルールとする．この操作により，グラマー記号列を伸張させてしまうような無駄なルールを含まないようにルール生成が行われる．また，偶然一致した Digram をルールとし，その後のルール生成に悪影響を及ぼす危険性もなくなる．

本研究では，中村らの手法と同様，最頻 Digram に基づいてグラマー生成を行うが，更に算術符号化を適用することを考える．このとき中村らの手法のように各 Digram の頻度が 1 になるまでルール生成を行うことが必ずしも圧縮率を上げることになるとは限らない．そこで，グラマー記号列 g に対して算術符号の理想符号長 $-\log_2 P(g)$(注3) を計算しながらグラマーの生成を行い，その値が最小となるグラマーを採用し符号化す

(注2)：非終端記号には全順序関係が定義されている．

(注3)：$P(g)$ はグラマー記号列 g の経験確率

る手法を提案する．提案手法の着眼点は，算術符号を適用するに当たって，符号長を最小にするグラマーを用いることによって高圧縮率の達成が期待されるという点にある．理想符号長を最小にするグラマーに対して，グラマー記号列を生成する際に，

(1) 全てのルールを連接させる手法 (3.1 節 (1))

(2) ルール S 内で他のルールを参照する手法 (3.1 節 (2))

を用い，算術符号を適用するものをそれぞれ提案手法 1，提案手法 2 とする．

4.2 理想符号長の計算

理想符号長の計算法について述べる．まず，グラマー記号列 g の経験確率 $P(g)$ を計算する確率モデルとして，本研究では各記号に i.i.d. を仮定する．なぜなら，頻度の多い Digram をそれぞれ非終端記号で置き換えることによってグラマー中のマルコフ性が薄れるからである．従って，グラマー記号列 g の経験確率 $P(g)$ は，

$$P(g) = \prod_{g_i \in \Sigma} \left(\frac{n(g_i)}{l_g} \right)^{n(g_i)} \qquad (1)$$

となる．ここで，l_g はグラマー記号列長，g_i はグラマー記号列内で出現する記号，$n(g_i)$ は記号 g_i のグラマー記号列内の出現頻度とする．よって，グラマー記号列 g を算術符号化したときの理想符号長は，

$$-\log_2 P(g) = -\log_2 \prod_{g_i \in \Sigma} \left(\frac{n(g_i)}{l_g} \right)^{n(g_i)} \qquad (2)$$

となる．

4.3 アルゴリズム

① データ系列 x をルール S の右辺として全て読み込み，各 Digram の出現頻度を求める．

② 各 Digram の出現頻度が全て 1 ならば ⑤ へ．そうでなければ最頻の Digram をルールとして登録し，その Digram をルールを表す非終端記号で置き換える．

③ それぞれの記号の出現頻度を調べ，理想符号長を式 (2) より計算する．

④ 置き換えた非終端記号も終端記号と同様に扱い，その直前および直後の記号との Digram の頻度を求め ② へ．

⑤ 理想符号長の最も小さなグラマーを採用し算術符号化する． □

5. 計算量及びメモリ量

本節では最頻 Digram を基にルールを生成する提案手法，及び中村らの手法 [1] が Sequitur アルゴリズムと同様，計算量及びメモリ量が $O(N)$ (N はデータ系列長) であることを示す．提案手法及び中村らの手法では以下の 3 つの操作がある．

(1) 最初に Digram の頻度を調べる．(この計算量を c_1 とする．)

(2) Digram を非終端記号で置き換える．(この計算量を c_2 とする．)

(3) グラマー記号列の理想符号長 $-\log_2 P(g)$ を計算する．(この計算量を c_3 とする．)

証明に用いる Notation を以下に示す．

< Notation >

N： 元データ系列の長さ．

$g^{(t)}$： t 時点(注4)のグラマー記号列．

g_j： グラマー記号列内で j 番目に出現する記号．

(注4)：t 時点とは t 個目のルールを生成した時点を指す．

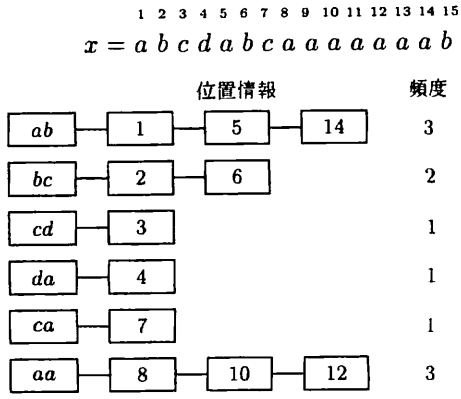

図 2 位置情報を保持するリスト構造

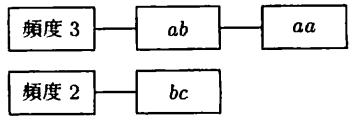

図 3 頻度ごとに分類されたリスト構造

$l_g^{(t)}$： t 時点のグラマー記号列の長さ．

$l_S^{(t)}$： t 時点のルール S の長さ．

$n^{(t)}(y)$： t 時点の記号 $y(\in \Sigma)$ のグラマー記号列中の頻度．

$\delta_k^{(t)}$： t 時点の最頻 Digram の $k(=1,2)$ 番目の記号．

$n_d^{(t)}$： t 時点の最頻の Digram の頻度．

$|R|$： 最終的に生成されたルール S 以外のルール数．

$|\Sigma_T|$： Σ_T の記号数．

[定理] 提案手法の計算量及びメモリ量は $O(N)$ である．

(証明) まず，最初に Digram の頻度を調べるためにルール S の右辺を見て，図 2 のリスト構造及び頻度情報を作成する計算量は $O(N)$ であるため，$c_1 = O(N)$ である．

次に，頻度ごとの Digram の分類に対する計算量であるが，最初に Digram の頻度を調べた後，図 3 に例を示すような各 Digram の出現頻度を保持するリスト構造を生成する．この手続きは登録すべき Digram の個数だけ計算が必要となる．Digram の数は高々 $|\Sigma_T|^2$ であるので，リスト構造生成の手続きは高々 $O(1)$ である．

次に，ルールが新たに生成されたことによるリスト構造の更新のために必要な総計算量 c_2 を考える．この手続きはルールの左辺の非終端記号で Digram を置き換える際に起こる．このとき 1 回の置き換えによってその Digram の前後の 2 つの Digram の頻度が減る．逆に置き換えた非終端記号の前後の 2 つの Digram が新しく生成される．結果的に 1 つの非終端記号の置き換えに必要な計算量はその Digram の頻度 $n_d^{(t)}$ に比例する．従って，最終的に $|R|$ 個の非終端記号について全て置き換える計算量 c_2 は各段階での最頻 Digram の頻度の合計

$$W_d = \sum_{t=1}^{|R|} n_d^{(t)}$$

に比例する．ここで，Digram を非終端記号へ 1 回置き換えるときにルール S の長さが 1 減ることから，

$$
\begin{aligned}
l_S^{(1)} &= N - n_d^{(1)} \\
l_S^{(2)} &= N - (n_d^{(1)} + n_d^{(2)}) \\
l_S^{(3)} &= N - (n_d^{(1)} + n_d^{(2)} + n_d^{(3)}) \\
&\vdots \\
l_S^{(|R|)} &= N - \sum_{t=1}^{|R|} n_d^{(t)} \\
&= N - W_d
\end{aligned}
$$

が成り立つ．ここで，$l_S^{|R|}$ は最終的なルール S の長さで非負なので，

$$
\begin{aligned}
l_S^{|R|} &= N - W_d \geqq 0 \\
W_d &\leqq N
\end{aligned}
$$

が成り立ち，Digram を非終端記号に置き換える総計算量は，$c_2 = O(N)$ となる．

最後に，理想符号長を計算する計算量であるが，グラマー記号列 $g^{(t)}$ の理想符号長 $L(g^{(t)})$ の値は以下の式によって求められる．式 (2) より，

$$
\begin{aligned}
L(g^{(t)}) &= -\log_2 P(g^{(t)}) \\
&= -\log_2 \prod_{g_j \in \Sigma} \left(\frac{n^{(t)}(g_j)}{l_g^{(t)}} \right)^{n^{(t)}(g_j)} \\
&= -\sum_{g_j \in \Sigma} n^{(t)}(g_j) \log_2 \left(\frac{n^{(t)}(g_j)}{l_g^{(t)}} \right) \\
&= \sum_{g_j \in \Sigma} n^{(t)}(g_j) \log_2 l_g^{(t)} - \sum_{g_j \in \Sigma} n^{(t)}(g_j) \log_2 n^{(t)}(g_j) \\
&= l_g^{(t)} \log_2 l_g^{(t)} - \sum_{g_j \in \Sigma} n^{(t)}(g_j) \log_2 n^{(t)}(g_j) \qquad (3)
\end{aligned}
$$

ここで，3.1 節 (2) ルール S 内で他のルールを参照する手法でグラマーを 1 次元化する場合には，

$$
l_g^{(t)} = l_g^{(t-1)} - n_d^{(t)} + 1 \qquad (4)
$$

$$
n^{(t)}(\delta_1^{(t)}) = n^{(t-1)}(\delta_1^{(t)}) - n_d^{(t)} + 1 \qquad (5)
$$

$$
n^{(t)}(\delta_2^{(t)}) = n^{(t-1)}(\delta_2^{(t)}) - n_d^{(t)} + 1 \qquad (6)
$$

が成り立つ．また新しく生成されたルールの左辺の非終端記号が $n_d^{(t)} - 1$ 個発生するので，結局 $L(g^{(t)})$ の値は，

$$
\begin{aligned}
L(g^{(t)}) &= L(g^{(t-1)}) \\
&\quad - l_g^{(t-1)} \log_2 l_g^{(t-1)} \\
&\quad + (l_g^{(t-1)} - n_d^{(t)} + 1) \log_2 (l_g^{(t-1)} - n_d^{(t)} + 1) \\
&\quad + n^{(t-1)}(\delta_1^{(t)}) \log_2 n^{(t-1)}(\delta_1^{(t)}) \\
&\quad - (n^{(t-1)}(\delta_1^{(t)}) - n_d^{(t)} + 1) \log_2 (n^{(t-1)}(\delta_1^{(t)}) - n_d^{(t)} + 1) \\
&\quad + n^{(t-1)}(\delta_2^{(t)}) \log_2 n^{(t-1)}(\delta_2^{(t)}) \\
&\quad - (n^{(t-1)}(\delta_2^{(t)}) - n_d^{(t)} + 1) \log_2 (n^{(t-1)}(\delta_2^{(t)}) - n_d^{(t)} + 1) \\
&\quad - (n_d^{(t)} - 1) \log_2 (n_d^{(t)} - 1) \qquad (7)
\end{aligned}
$$

によって更新できる．3.1 節 (1) 全てのルールを連接する手法についても同様にして示すことができる．このように，1 回非終端記号で置き換える際に理想符号長を更新する計算量は定数である．これは $|R|$ 回行われるので，理想符号長の計算に必要な総計算量 c_3 は高々 $O(N)$ となる．従って，このアルゴリズムに必要な総計算量は $c_1 + c_2 + c_3$ より $O(N)$ である．以上より，提案手法，中村らの手法の計算量は $O(N)$ であることが示された．

またメモリ量についても $O(N)$ であることは明らかであろう． □

以上より，提案手法，中村らの手法の計算量及びメモリ量は $O(N)$ であることが示された．

6. シミュレーション結果および考察

本節ではシミュレーションを行い，結果の考察を行う．

6.1 シミュレーション

様々なカルガリーデータ [8] を対象とし，Sequitur アルゴリズムおよび中村らの手法と，提案手法との圧縮率の比較を行う．情報源アルファベットは英数字である．

以下のように，グラマーの 1 次元化の仕方によって手法を分類し，シミュレーションを行う．それぞれの手法の設定をまとめたものを表 3 に示す．

- 設定 1　全てのルールを連接する手法によって一次元化する場合 (3.1 節 (1))

Sequitur と提案手法 1 の圧縮性能を比較する．これらの手法の大きな相違は，グラマーを生成する際に前方一致の Digram を基にするか，最頻の Digram を基にするかという点である．ここで，提案手法 1 では，Sequitur アルゴリズムと同様 Rule Utility を満たすようにルールの削除も行っている．更に提案手法の有効性を見るため，最頻 Digram に基づくが，全ての Digram の頻度が 1 になるまでルールを生成して算術符号化を適用する手法（手法 1 ）との比較も行った．圧縮率の結果を表 4 に示す．

- 設定 2　ルール S 内で他のルールを参照する手法によって一次元化する場合 (3.1 節 (2))

中村らの手法と提案手法 2 の圧縮性能を比較する．これらの手法の大きな相違は生成されたグラマーをそのまま符号語とするか，算術符号化するかという点である．この設定ではどちらもルールの削除は行わない．また，設定 1 と同様，提案手法の有効性を見るため，全ての Digram の頻度が 1 になるまでルールを生成して算術符号化を適用する手法（手法 2 ）との比較も行った．圧縮率の結果を表 5 に示す．

表 3　シミュレーションで比較した手法

設定	グラマー生成		ルール	符号化	手法
1	前方一致の Digram		連接	算術符号	Sequitur
	最頻 Digram	理想符号長最小			提案手法 1
		全ての頻度が 1			手法 1
2		理想符号長最小	内部参照		提案手法 2
		全ての頻度が 1			手法 2
				記号列そのまま	中村らの手法

6.2 シミュレーション結果

(1)　設定 1 では全てのデータにおいて，提案手法 1 の圧縮率は Sequitur に比べて改善されている．すなわち前方一致する Digram を基にルールを生成するよりも最頻の Digram に基づいてルール生成を行うことの有効性が確認された．また，提案手法 1 の圧縮率は手法 1 よりも改善されており理想符号長を計算しながらグラマー生成を行うことの有効性が現れている．

(2)　設定 2 では全てのデータにおいて，提案手法 2 の圧縮率は中村らの手法に比べ改善されている．これはグラマー記号列をそのまま符号語とするよりも算術符号化したほうが圧縮率が改善されることを示している．しかし，提案手法 2 の圧

表 4 シミュレーション結果 1

データ	サイズ [byte]	Sequitur [4]	提案手法 1	手法 1
bib	117,543	0.421	0.338	0.346
book1	785,394	0.447	0.379	0.401
book2	268,843	0.437	0.362	0.377
geo	102,400	0.746	0.636	0.621
news	387,170	0.488	0.415	0.419
obj1	21,504	0.674	0.596	0.621
obj2	246,814	0.492	0.427	0.433
paper1	54,413	0.515	0.428	0.439
paper2	83,932	0.488	0.401	0.420
paper3	47,628	0.546	0.450	0.473
paper4	15,582	0.433	0.455	0.476
paper5	12,276	0.628	0.544	0.563
paper6	39,126	0.538	0.451	0.460
pic	513,216	0.159	0.118	0.128
progc	41,100	0.504	0.428	0.434
progl	71,908	0.382	0.317	0.315
progp	51,347	0.361	0.304	0.296
trans	90,726	0.359	0.299	0.284

表 5 シミュレーション結果 2

データ	サイズ [byte]	中村らの手法	提案手法 2	手法 2	GZIP
bib	117,543	0.291	0.265	0.265	0.302
book1	785,394	0.390	0.331	0.331	0.402
book2	268,843	0.323	0.295	0.295	0.334
geo	102,400	0.715	0.604	0.604	0.669
news	387,170	0.372	0.328	0.328	0.378
obj1	21,504	0.576	0.499	0.499	0.483
obj2	246,814	0.389	0.334	0.334	0.329
paper1	54,413	0.377	0.331	0.331	0.346
paper2	83,932	0.363	0.324	0.324	0.359
paper3	47,628	0.418	0.360	0.360	0.386
paper4	15,582	0.392	0.359	0.359	0.367
paper5	12,276	0.457	0.420	0.420	0.419
paper6	39,126	0.403	0.346	0.346	0.345
pic	513,216	0.126	0.104	0.104	0.102
progc	41,100	0.379	0.327	0.327	0.327
progl	71,908	0.258	0.233	0.233	0.223
progp	51,347	0.250	0.220	0.220	0.221
trans	90,726	0.245	0.211	0.211	0.206

縮率と手法 2 の圧縮率が一致しており，理想符号長最小のグラマー生成による有効性は見られなかった．

(3) 設定 2 では，全般的に GZIP と比べてほぼ同等の圧縮率となっている．

6.3 考　　察

(1) 設定 1 では提案手法 1 は Sequitur に比べ，圧縮率が改善された．また手法 1 との比較により，理想符号長を計算する有効性があることが分かった．ここで，各データで採用されたグラマーの最頻 Digram の頻度は，4 が最も多かった．これにより，理想符号長を計算することの有効性が示せた理由としては，以下の点が挙げられる．

設定 1 では，全てのルールを連接する際に区切り記号を挿入するため，頻度が 3 の Digram をルールとして置き換えを行っても，グラマー記号列長は減らない．更に，頻度が 2 の Digram をルールとすれば，グラマー記号列長は増えてしまうことになる．これにより，頻度 4 の Digram が置き換えられている段階までは，符号長は減っていくが，それ以降は符号長が増えていくためルール生成により効率が悪くなると考えられる．

(2) 設定 2 では，提案手法 2 の方が中村らの手法よりも圧縮率が改善されたが，手法 2 との比較によりこれは算術符号の効果のみによって改善されたことが分かった．設定 2 では 3.1 節 (2) の手法によりルール S 内で他のルールを参照しているため，区切り記号が不必要である．従って同一の Digram が存在するならばルールとして置き換えを行えば行うほどグラマー記号列長が減り，理想符号長が単調に減少すると考えられる．このため，大部分のデータで途中のグラマーが採用されず，全ての Digram の頻度がほぼ 1 になったグラマーが採用されたのだと考えられる．

7. まとめと今後の課題

本研究では，最頻 Digram を基に，算術符号の理想符号長を計算しながら符号長の観点から最適なグラマーを生成する手法を提案した．

設定 1 についてはシミュレーションによる評価で提案手法の有効性を示すことができた．設定 2 では，理想符号長最小の基準を用いたことによる有効性を示すには至らなかったが，最頻 Digram 統合と算術符号を組み合わせる点での有効性は確認できた．更に提案手法及び中村らの手法の計算量及びメモリ量が Sequitur と同様に $O(N)$ であることを示した．

今後は最頻 Digram 統合を用いる圧縮法に対して圧縮率の理論的な評価を行い，ユニバーサル性を示したい．更にこの手法は GZIP に対してもほぼ同等な圧縮率を得ることができたので，更に有効なグラマー生成手法の提案を行いたい．

グラマー生成の手法を考える上で，今回は Digram のみを対象としているが，N-gram を考え，情報源に適した長さを採用するというアルゴリズムを考えることも可能である．また，Sequitur アルゴリズムはデータマイニング手法としても有効である．本手法で生成されたルールを見ると，多くの英単語がルールとなっている．そこで本手法をデータマイニングの分野へ適用することも考える予定である．

8. 謝　　辞

本研究を行うにあたり，数多くのご助言，ご支援を賜りました，早稲田大学 中澤真先生，八木秀樹氏，武蔵工業大学 後藤正幸先生，並びに早稲田大学平澤研究室の各氏に感謝いたします．

文　　献

[1] 中村 博文，村島 定行: "高頻度隣接の連結に基づいたデータ圧縮法," 第 14 回情報理論とその応用シンポジウム, pp.701-704, (1991).
[2] 中村 博文，村島 定行: "繰り返し現れる隣接文字の連結に基づくデータ圧縮法," 電子情報通信学会誌, Vol.J79-A, No.7 , pp.1319-1323, (1996).
[3] Craig G.Nevill-Mannning, Ian H.Witten: "Identifying Hierarchical Structure In Sequences a Linear-Time Algorithm," *Journal of Artificial Intelligence Research* 7, pp.67-82, (1997).
[4] Craig G.Nevill-Mannning, Ian H.Witten: "Compression and Explanation using Hierarchical Grammars," *The Computer Journal*, Vol.40, No.2/3, pp.103-116, (1997).
[5] J.C.Kieffer, E.Yang: "Grammar Based Codes: A New Class of Universal Lossless Source Code," *IEEE Trans. Inform. Theory*, Vol.46, No.3, pp.737-754, (2000).
[6] J.C.Kieffer, E.Yang, G.J.Nelson, P.Cosman: "Universal Lossless Compression Via Multilevel Pattern Matching," *IEEE Trans. Inform. Theory*, Vol.46, No.4, pp.1227-1245, (2000).
[7] 山本 博資: "ユニバーサルデータ圧縮アルゴリズムの変遷—基礎から最新手法まで—," 2001 年情報論的学習理論ワークショップ予稿集, pp.339-348, (2001)
[8] ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression. corpus/
[9] 植松 友彦: 文書データ圧縮アルゴリズム入門, CQ 出版社, (1994).
[10] 韓 太舜，小林欣吾: 情報と符号化の数理, 培風館, (1999).
[11] 北 研二: 確率的言語モデル, 東京大学出版会, (1999).